

Natural Language Processing

Lab 5: Word Embeddings and Similarity

Objective

This lab aims to familiarize you with semantic similarity through vector representations of language. You'll explore how words and texts can be compared quantitatively, which is foundational in natural language processing (NLP) for tasks like search or recommendation systems.

Introduction

Word embeddings represent words as dense vectors in a continuous vector space, capturing semantic relationships based on their usage in text. This lab builds on the distributional hypothesis, which posits that words occurring in similar contexts tend to have similar meanings. By working with pre-trained models, you'll compute similarities using cosine similarity, a metric that evaluates the cosine of the angle between vectors to determine their relatedness, ranging from -1 (opposite) to 1 (identical). This approach is widely used in NLP for applications like sentiment analysis, information retrieval, and machine translation.

The lab emphasizes hands-on computation with spaCy, a robust NLP library that provides efficient access to word vectors through models like `en_core_web_md` (medium, with 20k vectors) or `en_core_web_lg` (large, with 685k vectors). These models include static embeddings, where each word has a fixed vector, and support for context-sensitive representations via transformer-based pipelines.

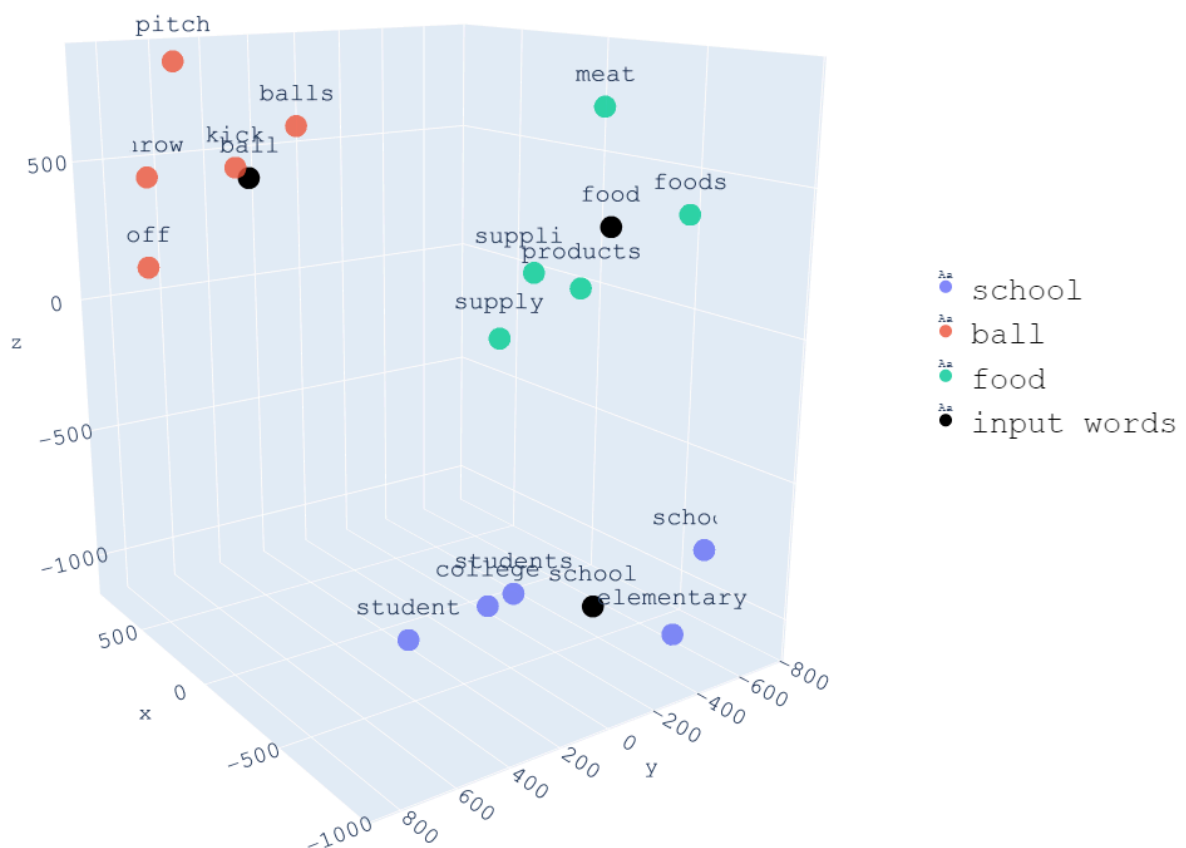
Word Vectors

A word vector is basically a dense representation of a word. What's surprising about these vectors is that semantically similar words have similar word vectors. Word vectors are great for semantic similarity applications, such as calculating the similarity between words, phrases, sentences, and documents.

Word vectors are produced by algorithms that make use of the fact that similar words appear in similar contexts. To capture the meaning of a word, a word vector algorithm collects information about the surrounding words that the target word appears with. This paradigm of capturing semantics for words by their surrounding words is called distributional semantics.

Distributional Semantics

Distributional semantics infers word meanings from statistical patterns in text, relying on vector space models (VSMs) where words are points in high-dimensional space. Semantic similarity is quantified using metrics like cosine similarity or Euclidean distance. This framework enables capturing relationships such as synonyms or hyponyms but may include noise like antonyms. Advanced models like BERT provide contextualized embeddings, adapting vectors based on surrounding text.



```
!python -m spacy download en_core_web_lg
```

```
import spacy
nlp = spacy.load("en_core_web_lg")
doc = nlp("I ate a banana.")
print(doc[3].vector)
```

```
print(type(doc[3].vector))
print(doc[3].vector.shape)
```

The Doc and Span objects also have vectors. The vector of a sentence or a span is the average of its words' vectors.

```
print(doc[1:3].vector)
```

Only the words in the model's vocabulary have vectors; words that are not in the vocabulary are called OOV (out-of-vocabulary) words. `token.is_oov` and `token.has_vector` are two methods we can use to query whether a token is in the model's vocabulary and has a word vector.

```
doc = nlp("You went there afskfsd.")
for token in doc:
    print("Token is:", token, ", OOV:", token.is_oov, ", Token has
vector:", token.has_vector)
```

Lab Task 1

Use The Adventures of Sherlock Holmes to select two "presentative" texts from this detective story:

1. Read the Adventures_Holmes.txt text file.
2. Save the content into a string object "holmes_doc".
3. Plot the Semantic Graphs for these two texts.
4. Perform the Similarity text for these two documents. See what you found.

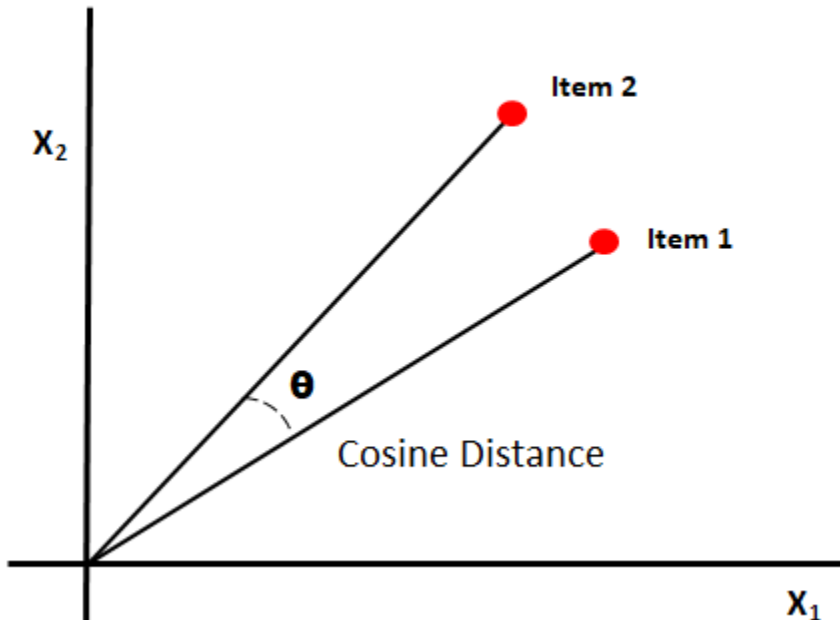
Cosine Similarity

Cosine similarity measures the alignment between two vectors A and B as:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \cdot \sqrt{\sum_{i=1}^n B_i^2}}$$

where the dot product is divided by the product of their norms. In NLP, this is preferred over Euclidean distance as it normalizes for vector magnitude, focusing on direction—ideal for comparing texts of varying lengths. For example, "doctor" and "nurse" often have high similarity due to co-occurrence in medical contexts, while "doctor" and "car" do not.

Cosine Distance/Similarity



Lab Task 2

Write a program, without using any library, for calculating the cosine similarity between:

1. Two words.
2. Two sentences.

Cosine Similarity with spaCy

In order to calculate the cosine similarity with spaCy, we can use the `similarity()` method of the `Doc` object.

```
doc0 = nlp("I like apples.")
doc1 = nlp("Apples keep the doctors away.")
print(doc0[2].similarity(doc1[3]))
print(doc0.similarity(doc1))
```

Lab Task 3

Perform lab task 2 while using spaCy and compare the results in terms of values and code execution time.